Politechnika Krakowska

Wydział Inżynierii Elektrycznej i Komputerowej

Informatyka, rok IV, niestacjonarne

Projekt z przedmiotu
Sztuczna Inteligencja

„Przewidywanie popularności artykułów (Online News Popularity).”

Autorzy:
Arkadiusz Pajor
Mateusz Ornat

Spis treści

1. Opis projektu i wykorzystanego środowiska

Projekt polega na utworzeniu modelu uczenia maszynowego, który będzie przewidywał na podstawie przyjętych atrybutów, jak często udostępniane są artykuły publikowane w Internecie.

Projekt utworzony jest w języku Python z wykorzystaniem biblioteki sklearn, która zajmuje się uczeniem maszynowym. Wykorzystano tutaj również notatniki Jupytera, które wprowadzają większą czytelność i intuicyjność. W prosty sposób pozwalają na zaprojektowanie modeli, a także na szybki dostęp do wyników.

Link do repozytorium na github: https://github.com/Arekaaa/OnlineNewsPopularity

1.1. Wykorzystane modele uczące

Zastosowano następujące modele uczące:

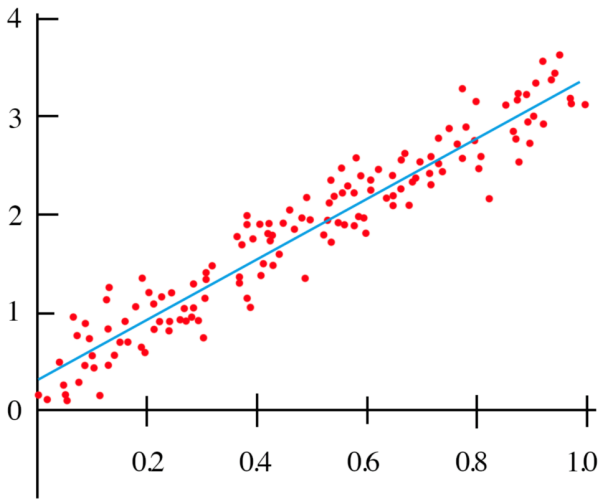
- Regresja liniowa,
- Regresja liniowa z zastosowaniem redukcji wymiarowości metodą PCA i TruncatedSVD,
- Sieć neuronowa,
- Sieć neuronowa z zastosowaniem redukcji wymiarowości metodą PCA i TruncatedSVD.

1.2. Regresja liniowa

Regresja, model regresyjny jest jedną z najbardziej popularnych metod analizy danych statystycznych. Główną ideą regresji jest przewidywanie, prognozowanie danych dla pewnej zmiennej na podstawie innych zmiennych. Innymi słowy, jaką wartość przyjmie dana zmienna gdy, będziemy znali wartość innej zmiennej. Oczywiście, aby móc "poszukiwać" wartości jednej zmiennej na podstawie innej zmiennej, musimy za pomocą analizy regresji skonstruować model regresyjny. Model, który będzie z założonym błędem statystycznym przewidywał wartość, poziom danej cechy.

Regresja liniowa jest najprostszym wariantem regresji w statystyce. Zakłada ona, że zależność pomiędzy zmienną objaśnianą, a objaśniająca jest zależnością liniową. Tak jak w analizie korelacji, jeżeli jedna wartość wzrasta to druga wzrasta (dodatnia korelacji) lub spada (korelacja ujemna). W regresji liniowej zakłada się, że wzrostowi jednej zmiennej (predyktor, predyktory) towarzyszy wzrost lub spadek na drugiej zmiennej. Co więcej, nazwa

regresji liniowej odnosi się, że funkcja regresji przyjmuje postać funkcji liniowej, czyli **y = bx+a.**



Rysunek 1. Przykładowy wykres regresji liniowej.

Analiza regresji liniowej ma na celu wyliczenie takich współczynników regresji (współczynników w modelu liniowym), aby model jak najlepiej przewidywał wartość zmiennej zależnej, aby błąd oszacowania był jak najmniejszy. Tak więc analiza regresji "dopasowuje" taką linię prostą do badanych (liniowa zależność), aby model był jak najlepszy (obarczony jak najmniejszym błędem losowym).

1.3. Redukcja wymiarowości

W projekcie zastosowano dwa rodzaje redukcji wymiarowości w celu porównania ich wyników:

- Metoda PCA (Principal component analysis) jest to liniowa redukcja wymiarów za pomocą dekompozycji wartości pojedynczej danych w celu rzutowania ich na przestrzeń o niższych wymiarach. Dane wejściowe są wyśrodkowane, ale nie są przeskalowane.

- Metoda TruncatedSVD (Singular value decomposition) jest to metoda, która dokonuje liniowej redukcji wymiarowości za pomocą skróconego rozkładu wartości pojedynczej. W przeciwieństwie do PCA, estymator nie centruje danych przed obliczeniem rozkładu wartości osobliwych. Oznacza to, że może efektywnie pracować z rzadkimi macierzami.

1.4. Sieć neuronowa

Sieci neuronowe, są to struktury składające się z neuronów połączonych synapsami. Sztuczne sieci neuronowe składają się z trzech typów warstw:

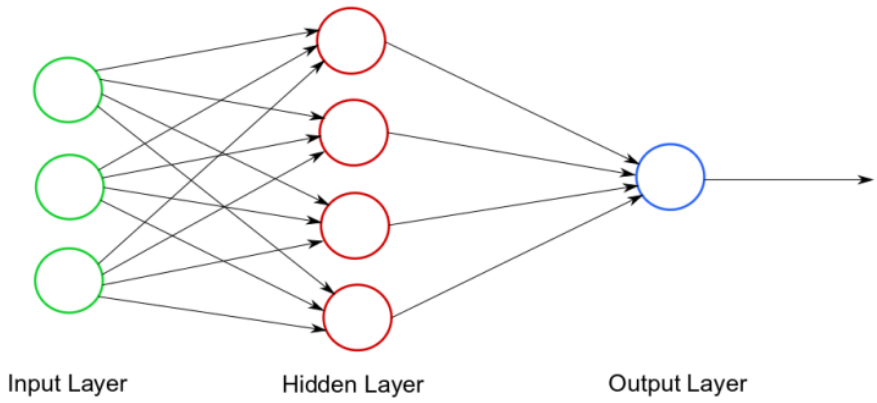
- wejściowej (zbiera dane i przekazuje je dalej)
- ukrytej (tu szukane są powiązania między neuronami, czyli zachodzi proces uczenia się)
- wyjściowej (gromadzi wnioski, wyniki analizy)

Sieć neuronowa może składać się z dowolnej liczby warstw. W technologii informacyjnej (IT) sieć neuronowa to sprzęt albo oprogramowanie (może być jedno i drugie) wzorowane na działaniu neuronów w ludzkim mózgu.

Zazwyczaj sieć neuronową tworzy wiele warstw:

- Do pierwszej warstwy – analogicznie jak w przypadku obrazów rejestrowanych np. przez nerwy wzrokowe u człowieka – trafiają nieprzetworzone dane wejściowe
- Każda kolejna warstwa otrzymuje dane będące wynikiem przetworzenia danych w warstwie poprzedniej
- To, co wytwarza ostatnia warstwa, to tzw. dane wyjściowe systemu

Sieć neuronowa funkcjonuje jak ludzki mózg: każdy neuron przeprowadza własne proste obliczenia, a sieć, którą tworzą wszystkie neurony, zwielokrotnia potencjał tych obliczeń. Sieci neuronowe wykorzystywane w sztucznej inteligencji są zorganizowane na tej samej zasadzie – ale z jednym wyjątkiem: by wykonać określone zadanie, połączenia między neuronami można odpowiednio dostosować. Technologia sieci neuronowych ma wiele praktycznych zastosowań. Używa się jej m.in. do rozpoznawania pisma ręcznego w celu przetwarzania czeków, transkrypcji mowy na tekst, prognozowania pogody czy rozpoznawania twarzy.



Rysunek 2. Schemat budowy sieci neuronowej.

2. Realizacja projektu

2.1. Opis i analiza danych

Projekt bazuje na repozytorium „Online News Popularity – zestaw danych dostarczany przez UCI ML Dataset (Machine Learning Repository). Zawiera szczegółowe informacje na temat 39 000 artykułów opublikowanych w 'Mashables' oraz liczbę udostępnień każdego z nich. Dane te wykorzystane zostały w celu zbudowania modelu do przewidywania popularności artykułów.

Zestaw danych możemy pobrać z repozytorium na stronie:
https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity#

Baza składa się :

- **Liczba instancji**: 39797
- **Liczba atrybutów**: 61 (58 atrybutów predykcyjnych, 2 nieprzewidywalne, 1 pole bramkowe)

- Informacje o atrybutach:
  - **URl:** adres artykułów
  - **time delta:** dni miedzy publikacją artykułu a pozyskiwanie zestawu danych
  - **n_tokens_tittle:** liczba słów w tytule
  - **n_tokens_content:** liczba słów w treści
  - **n_unique_tokens:** współczynnik niepowtarzalnych słów w treści
  - **n_non_stop_words:** współczynnik nieprzerwanych słów w treści
  - **n_non_stop_unique_tokens:** współ. niepowt. słów non-stop w treści
  - **num_hrefs:** liczba linków
  - **num_self_hrefs:** liczba linków do innych artykułów opublikowanych przez Mashable
  - **num_imgs:** liczba zdjęć
  - **num_videos:** liczba filmów
  - **average_token_length:** Średnia długość słów w zadowolony
  - **num_keywords:** liczba słów kluczowych w metadanych
  - **data_channel_is_lifestyle:** Czy kanał danych „Lifestyle"?
  - **data_channel_is_entertainment:** Czy kanał danych „Entertainment"?
  - **data_channel_is_bus:** Czy kanał danych „Business"?
  - **data_channel_is_socmed:** Czy kanał danych „Social Media"?
  - **data_channel_is_tech:** Czy kanał danych „Tech"?
  - **data_channel_is_world:** Czy kanał danych „World"?
  - **kw_min_min:** Najgorsze słowo kluczowe (min. Udostępnienia)
  - **kw_max_min:** Najgorsze słowo kluczowe (maks. Akcje)
  - **kw_avg_min:** Najgorsze słowo kluczowe (śr. Akcje)
  - **kw_min_max:** Najlepsze słowo kluczowe (min. Akcje)
  - **kw_max_max:** Najlepsze słowo kluczowe (maks. Akcje)

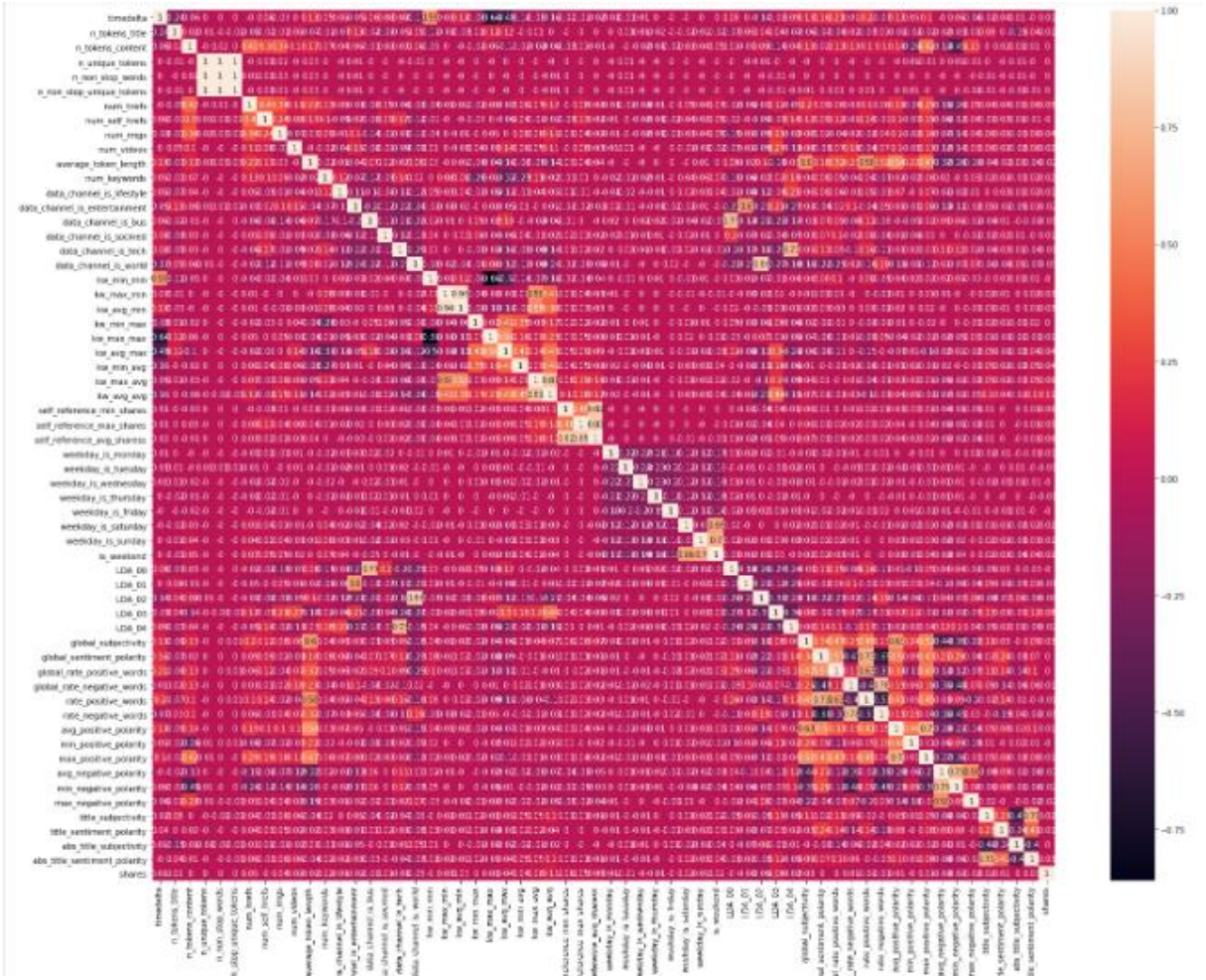
- **kw_avg_max:** Najlepsze słowo kluczowe (śr. Akcje)
- **kw_min_avg:** Śr. słowo kluczowe (min. udostępnienia)
- **kw_max_avg:** Śr. słowo kluczowe (maks. udostępnienia)
- **kw_avg_avg:** Śr. słowo kluczowe (śr. akcje)
- **self_reference_min_shares:** min. udziały przywoływanych artykułów w M.
- **self_reference_max_shares:** Max. udziały przywoływanych artykułów w M.
- **elf_reference_avg_sharess:** Śr. udziały przywoływanych artykułów w M.
- **weekday_is_monday:** Czy artykuł został opublikowany w poniedziałek?
- **weekday_is_tuesday:** Czy artykuł został opublikowany we wtorek?
- **weekday_is_wednesday:** Czy artykuł został opublikowany w środę?
- **weekday_is_thursday**: Czy artykuł został opublikowany w czwartek?
- **weekday_is_friday:** Czy artykuł został opublikowany w piątek?
- **weekday_is_saturday**: Czy artykuł został opublikowany w sobotę?
- **weekday_is_sunday:** Czy artykuł został opublikowany w niedzielę?
- **is_weekend:** Czy artykuł został opublikowany w weekend?
- **LDA_00:** Bliskość tematu LDA 0
- **LDA_01:** Bliskość tematu 1 LDA
- **LDA_02**: Bliskość tematu 2 LDA
- **LDA_03:** Bliskość tematu 3 LDA
- **LDA_04:** Bliskość tematu LDA 4
- **global_subjectivity:** Subiektywność tekstu
- **global_sentiment_polarity:** Polaryzacja sentymentu tekstowego
- **global_rate_positive_words:** Współczynnik pozytywnych słów w treści
- **global_rate_negative_words:** Współczynnik wykluczających słów w treści
- **rate_positive_words:** Współczynnik pozytywnych słów wśród
- nieneutralnych
- **rate_negative_words:** Współczynnik negatywnych słów wśród
- nieneutralnych
- **avg_positive_polarity:** Śr. polaryzacja pozytywnych słów
- **min_pozytywna_polaryzacja:** min. polaryzacja pozytywnych słów
- **max_positive_polarity:** Max. polaryzacja pozytywnych słów
- **avg_negative_polarity:** Śr. polaryzacja negatywnych słów
- **min_negative_polarity:** Min. polaryzacja negatywnych słów
- **max_negative_polarity:** Max. polaryzacja negatywnych słów
- **title_subjectivity:** Subiektywność tytułu
- **title_sentiment_polarity:** Polaryzacja tytułu
- **abs_title_subjectivity:** Absolutny poziom podmiotowości
- **abs_title_sentiment_polarity:** Absolutny poziom polaryzacji
- **shares:** Liczba udostępnień (target)

	url	timedelta	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unique_tokens
0	http://mashable.com/2013/01/07/amazon-instant-...	731.0	12.0	219.0	0.663594	1.0	0.815385
1	http://mashable.com/2013/01/07/ap-samsung-spon...	731.0	9.0	255.0	0.604743	1.0	0.791946
2	http://mashable.com/2013/01/07/apple-40-billio...	731.0	9.0	211.0	0.575130	1.0	0.663866
3	http://mashable.com/2013/01/07/astronaut-notre...	731.0	9.0	531.0	0.503788	1.0	0.665635
4	http://mashable.com/2013/01/07/att-u-verse-apps/	731.0	13.0	1072.0	0.415646	1.0	0.540890

5 rows × 61 columns

Rysunek 3. Część zestawu danych z repozytorium 'Online News Popularity'.

Poprzez stworzenie macierzy korelacji, możemy zauważyć zależności liniowe między zmiennymi danego zestawu danych. Poniższy rysunek przedstawia te zależności:



Rysunek 4. Macierz korelacji zmiennych w zestawie danych 'Online News Popularity'.

Żądanie cytowania:

K. Fernandes, P. Vinagre i P. Cortez. Proaktywny inteligentny system wspomagania decyzji w zakresie przewidywania popularności artykułów online. Materiały z 17. EPIA 2015 - Portugalska konferencja na temat sztucznej inteligencji, wrzesień, Coimbra, Portugalia.

2.2. Optymalizacja danych

Aby wytrenować model przeprowadziliśmy następujące operacje na danych:

- Sprawdzenie pustych wartości.

  Aby móc zacząć pracować nad wytrenowaniem modelu uczenia maszynowego ważnym krokiem jest sprawdzenie pustych wartości w bazie danych.
  Można tego dokonać za pomocą prostej w budowie funkcji przedstawionej na poniższym zdjęciu.

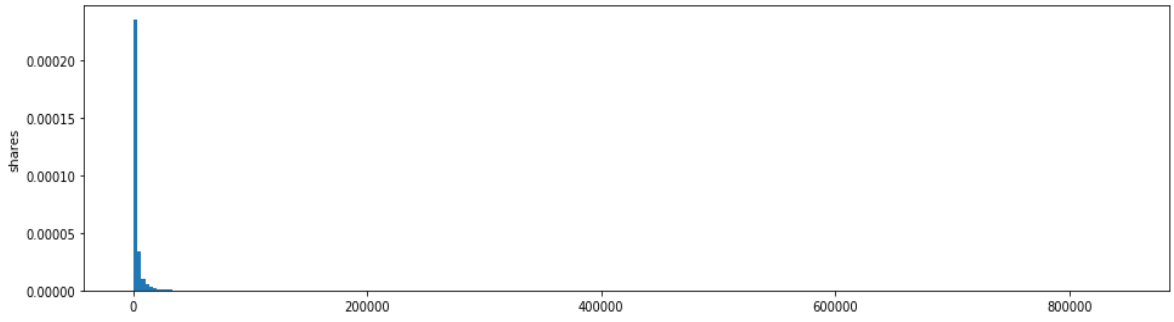
```
In [4]: #SPRAWDZENIE PUSTYCH WARTOŚCI
        news_data.isnull().sum()

Out[4]: url                             0
        timedelta                       0
        n_tokens_title                  0
        n_tokens_content                0
        n_unique_tokens                 0
                                       ..
        title_subjectivity              0
        title_sentiment_polarity        0
        abs_title_subjectivity          0
        abs_title_sentiment_polarity    0
        shares                          0
        Length: 61, dtype: int64
```

Rysunek 5. Sprawdzenie pustych wartości.

  Jak widać funkcja nie wykryła pustych wartości.

- Usunięcie wartości odstających.

  Aby zwiększyć dokładność modelu niezbędne jest usunięcie wartości odstających. Jest to usunięcie wartości, które zbyt mocno odbiegają od atrybutu shares, czyli są bardzo nisko skorelowane.
  Wykres przed zastosowaniem algorytmu:



Rysunek 6. Wartości odstające.

Algorytm usuwania wartości odstających:

```
In [8]: #Usuwanie wartości odstających
        Q1 = news_data[' shares'].quantile(0.25)
        Q3 = news_data[' shares'].quantile(0.75)
        IQR = Q3 - Q1
        LTV= Q1 - (1.5 * IQR)
        UTV= Q3 + (1.5 * IQR)
        fixedData = news_data.drop(news_data[news_data[' shares'] > UTV].index)
        fixedData.shape

Out[8]: (35103, 60)
```

Rysunek 7. Implementacja - Usuwanie wartości odstających.

Powyższy algorytm wyszukuje wartości, które są wyjątkowo nisko skorelowane z atrybutem shares. Wykorzystuje definicję kwantylu, a także różnicę między trzecim a pierwszym kwantylem. Następnie usuwa ze zbioru danych wartości nisko skorelowane z naszym targetem.

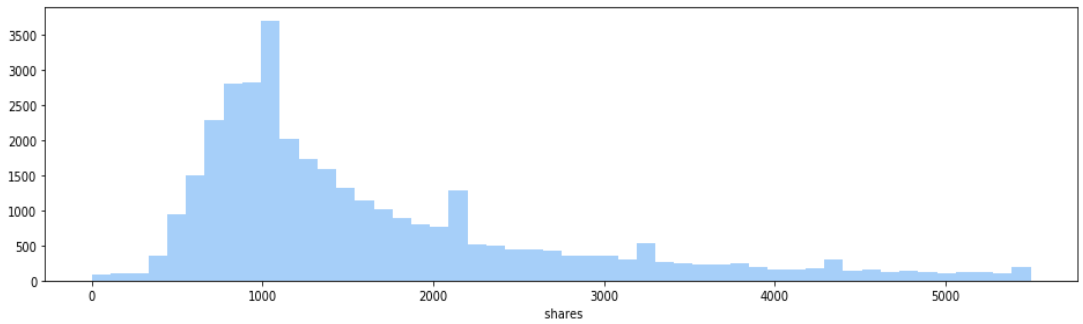
Po zastosowaniu algorytmu dane prezentują się następująco:

```
In [24]: print(f'Dane przed usunięciem wartości odstających = {news_data.shape}')
         print(f'Dane po usunięciu wartości odstających = {fixedData.shape}')
         print(f'Liczba wartości odstających = {news_data.shape[0] - fixedData.shape[0]}')

         Dane przed usunięciem wartości odstających = (39644, 60)
         Dane po usunięciu wartości odstających = (35103, 60)
         Liczba wartości odstających = 4541
```

Rysunek 8. Różnica po zastosowaniu algorytmu.

Z powyższego rysunku wynika, że algorytm usunął 4541 nisko skorelowanych wartości. Wykres przedstawiający nasz target po zastosowaniu algorytmu jest zobrazowany poniżej:

```
In [30]: plt.subplots(figsize=(16,4))
         sns.distplot(fixedData[' shares'], hist=True, kde=False, color='#2289F0')

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x264d97ab088>
```



Rysunek 9. Różnica po zastosowaniu algorytmu.

- Standaryzacja danych

  Różne atrybuty mają szeroki zakres wartości, co może niekorzystnie wpływać na predykcję modelu. Aby tego uniknąć stosuje się standaryzacje danych. Biblioteka sklearn zapewnia różne rodzaje takich operacji. W projekcie zastosowana została funkcja MinMaxScaler. Funkcja ta skaluje każdy atrybut osobno, tak aby znajdował się on w bardziej korzystnym zestawie, tj. między zero a jeden.

```
In [15]: # STANDARYZACJA
         from sklearn.preprocessing import MinMaxScaler

         scaler = MinMaxScaler()
         data = scaler.fit_transform(data)
         data

Out[15]: array([[0.38888889, 0.03284389, 0.6047431 , ..., 0.5       , 1.        ,
                  0.        ],
                 [0.38888889, 0.02717671, 0.57512955, ..., 0.5       , 1.        ,
                  0.        ],
                 [0.38888889, 0.06839258, 0.5037879 , ..., 0.5       , 1.        ,
                  0.        ],
                 ...,
                 [0.44444444, 0.05692942, 0.51635516, ..., 0.56818182, 0.09090909,
                  0.13636364],
                 [0.22222222, 0.08784132, 0.53949331, ..., 0.5       , 1.        ,
                  0.        ],
                 [0.44444444, 0.02022154, 0.70198678, ..., 0.625     , 0.33333333,
                  0.25      ]])
```

Rysunek 10. Standaryzacja danych.

- Usunięcie wartości nie biorących udziału w procesie predykcji.

  Wykluczono dwa atrybuty, które nie są potrzebne do trenowania modelu. Są to „url” oraz „timedelta”. Usunięty został również atrybut „shares”. Jest to wartość oczekiwana, którą należy przewidzieć, więc nie umieszczamy jej w danych wejściowych.
  Dzięki temu z 61 kolumn, otrzymano 58 kolumn dla danych wejściowych, natomiast dla wartości oczekiwanej przypisano tylko jedną kolumnę o nazwie „shares”.
  Cały proces w praktycznym podejściu wygląda następująco:

```
In [6]: news_data = news_data.drop(['url'], axis=1)
```

```
In [14]: #USTALENIE DANYCH WEJŚCIOWYCH I TARGETU
         data = fixedData.iloc[1:,1:-1]
         target = fixedData.iloc[1:,-1]

         data.head()
```

Out[14]:

	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	ave
1	9.0	255.0	0.604743	1.0	0.791946	3.0	1.0	1.0	0.0	
2	9.0	211.0	0.575130	1.0	0.663866	3.0	1.0	1.0	0.0	
3	9.0	531.0	0.503788	1.0	0.665635	9.0	0.0	1.0	0.0	
4	13.0	1072.0	0.415646	1.0	0.540890	19.0	19.0	20.0	0.0	
5	10.0	370.0	0.559889	1.0	0.698198	2.0	2.0	0.0	0.0	

5 rows × 58 columns

```
In [15]: target.head()

Out[15]: 1     711
         2    1500
         3    1200
         4     505
         5     855
         Name:  shares, dtype: int64
```

Rysunek 11. Usunięcie wartości niezdatnych do predykcji.

2.3. Sposób trenowania modeli

W projekcie zostały wyszkolone cztery różne modele uczenia maszynowego. W poniższych rozdziałach znajduje się dokładny opis szkolenia tychże modeli wraz z ich wynikami.

2.3.1. Trenowanie modelu – Regresja liniowa

Po optymalizacji danych w tym przypadku , krokach podjętych w rozdziale 2.2 – tj. sprawdzenie pustych wartości, stworzenie macierzy korelacji, wykonanie standaryzacji danych oraz usunięcie wartości odstających, możemy przystąpić do trenowania naszego modelu. Porównując dane załadowane bezpośrednio z repozytorium MCI, a dane po optymalizacji zestawu, widzimy jak zmienia się rozmiar ‘datasetu’. Różnica wynosi około 5000 danych mniej.

```
In [2]: news_data.shape
Out[2]: (39644, 61)
```

Rysunek 12. Rozmiar zestawu danych przed optymalizacją.

```
Training dataset:
data_train: (31591, 58)
target_train: (31591,)

Testing dataset:
data_test: (3511, 58)
target_test: (3511,)
```

Rysunek 13. Rozmiar zestawu danych po optymalizacji.

W pierwszym etapie tuż przed standaryzacją danych, ustalamy nasze dane wejściowe oraz target, czyli cel trenowanego modelu regresji liniowej.

```
#USTALENIE DANYCH WEJŚCIOWYCH I TARGETU
data = fixedData.iloc[1:,1:-1]
target = fixedData.iloc[1:,-1]

data.head()
```

	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	ave
1	9.0	255.0	0.604743	1.0	0.791946	3.0	1.0	1.0	0.0	
2	9.0	211.0	0.575130	1.0	0.663866	3.0	1.0	1.0	0.0	
3	9.0	531.0	0.503788	1.0	0.665635	9.0	0.0	1.0	0.0	
4	13.0	1072.0	0.415646	1.0	0.540890	19.0	19.0	20.0	0.0	
5	10.0	370.0	0.559889	1.0	0.698198	2.0	2.0	0.0	0.0	

5 rows × 58 columns

Rysunek 14. Implementacja - ustalenie danych wejściowych oraz targetu.

Możemy zauważyć, że w naszym modelu pomijamy atrybuty ‘url’, ‘timedelta’ oraz ‘shares’ w celu uzyskania możliwie jak najdokładniejszego modelu. Atrybuty wyżej wymienione nie są używane w procesie uczenia modelu, są zbędne w tym procesie.

Tym samym zmienia się rozmiar naszej tabeli z 5 x 61 na 5x 58. Po wykonaniu standaryzacji danych, możemy przejść do podzielenia danych na modele uczące i testowe.

```
#DZIELENIE DANYCH NA MODELE UCZĄCE I TESTOWE

from sklearn.model_selection import train_test_split
data_train, data_test, target_train, target_test = train_test_split(data, target, test_size = 0.10)
print("Training dataset:")
print("data_train:", data_train.shape)
print("target_train:", target_train.shape)
print("\nTesting dataset:")
print("data_test:", data_test.shape)
print("target_test:", target_test.shape)
```

```
Training dataset:
data_train: (31591, 58)
target_train: (31591,)

Testing dataset:
data_test: (3511, 58)
target_test: (3511,)
```

Rysunek 15. Implementacja - dzielenie danych na modele uczące i testowe.

Dane uczące wynoszą 90% zestawu, natomiast dane testowe 10%. Wielkość danych testowych określamy atrybutem ‘test_size=0.10’. W następnym kroku rozpoczynamy uczenie modelu.

```
#UCZENIE MODELU
from sklearn.linear_model import LinearRegression

linear_regression = LinearRegression()
linear_regression.fit(data_train, target_train)
```

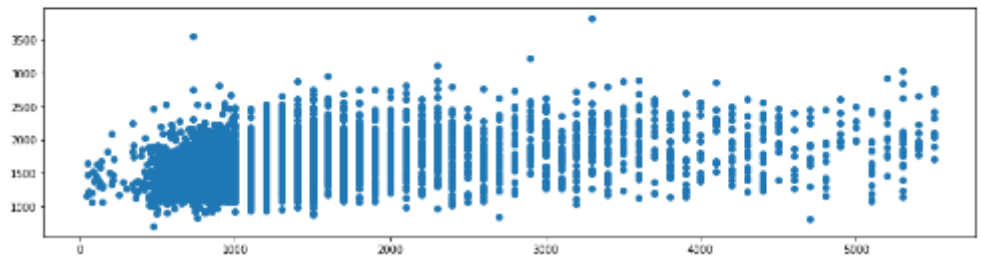
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Rysunek 16. Implementacja - uczenie modelu.

Korzystając z funkcji ‘linear_regression.predict(data_test) przewidujemy ilość udostępnień artykułów. Wykres predykcji przedstawiony na poniższym rysunku przedstawia wyszkolony model.

```
#WYKRES PREDYKCJI
plt.scatter(target_test, prediction_linear)
print("Wykres predykcji")
```

```
Wykres predykcji
```



Rysunek 17. Wykres predykcji regresji liniowej wytrenowanego modelu.

Jak widać na powyższym rysunku wykres nie przypomina idealnego wykresu regresji liniowej przedstawionego na rysunku numer 1. Oznacza to, że model posiada niski wynik wytrenowania.

Do estymowania wartości a i b używa się metody najmniejszych kwadratów. Zakłada ona, że będziemy szukać minimum dla sumy kwadratów różnic wartości obserwowanych i wartości teoretycznej (obliczonej z naszego równania).

```
#Ocena jakości metodą średniego błędu kwadratowego
from sklearn.metrics import mean_squared_error
mse =  mean_squared_error(target_test, prediction_linear)
rmse = np.sqrt(mse)
print("Średni błąd kwadratu wyuczonego modelu: %.2f" %  mse)
print("Pierwiastek średniego błędu kwadratowego: %.2f" %  rmse)
```

```
Średni błąd kwadratu wyuczonego modelu: 1055258.01
Pierwiastek średniego błędu kwadratowego: 1027.26
```

Rysunek 18. Implementacja - ocena jakości średniego błędu kwadratowego.

W celu określenia wyniku wariancji wytrenowanego modelu używamy metody ‘r2_score’.

```
from sklearn.metrics import r2_score
print('Variance score: %.2f' % r2_score(target_test, linear_regression.predict(data_test)))
```

```
Variance score: 0.12
```

Rysunek 19. Wynik wariancji wytrenowanego modelu.

Wynik wariancji wynosi zaledwie 12 %, oznacza to że model jest słabo wytrenowany. Pomimo przeprowadzenia wielu optymalizacji na danych wynik jest niski. Może to oznaczać, że dane z naszego ‘datasetu’ są problematyczne w procesie trenowania.

2.3.2. Trenowanie modelu – Regresja liniowa z redukcją wymiarowości

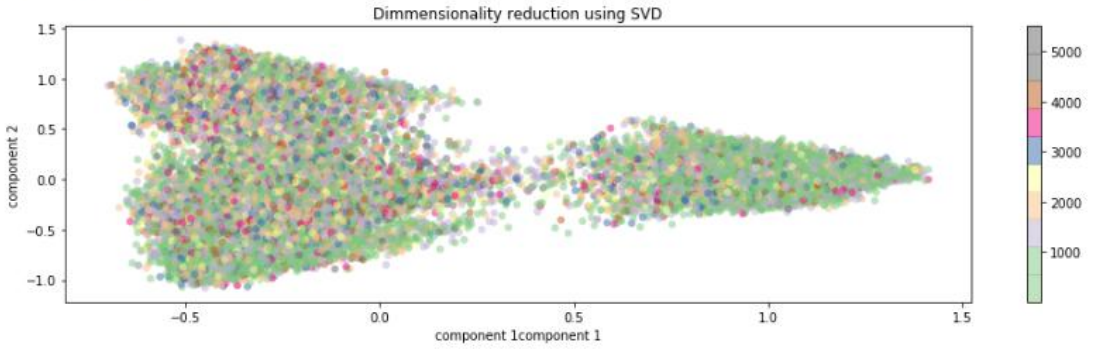
Proces trenowania modelu regresji liniowej z redukcją wymiarowości bazuje na modelu zaprezentowanym w podrozdziale 2.3.2. Do momentu standaryzacji danych, proces szkolenia przebiega identycznie, później jednak używamy metody PCA do zastosowania redukcji wymiarowości. Obrazuje to poniższy fragment kodu.

```
In [43]: #-------------------------------------------------------------------------------
         # Regresja liniowa z usuniętymi wartościami odstającymi z zastosowaniem SVD redukcji wymiarowości
         from sklearn.decomposition import PCA

         pca = PCA(n_components=40) #Metoda PCA
         pca_data = pca.fit(data).transform(data)
         pca_data

Out[43]: array([[-0.3555699 ,  0.00866528,  1.23732807, ...,  0.00342305,
                   0.01260055,  0.01503904],
                 [-0.41505111,  0.34104306,  0.74493274, ..., -0.00501035,
                  -0.0094847 ,  0.01465764],
                 [ 0.03223653, -0.39460297, -0.28554368, ...,  0.01140349,
                  -0.01669401, -0.00609484],
                 ...,
                 [-0.227935  , -0.34363471, -0.17989547, ...,  0.00802174,
                   0.00784164,  0.01420244],
                 [ 1.26219197,  0.09502245,  0.18772458, ..., -0.00180843,
                   0.00843977,  0.01342378],
                 [-0.45036292, -0.71759545, -0.38607578, ...,  0.07241938,
                   0.02678224, -0.04323739]])
```

Rysunek 20. Dekompozycja wartości pojedynczej danych w celu rzutowania ich na przestrzeń o niższych wymiarach.

Wizualizacja danych na wykresie:

```
In [44]: #Wizualizacja na wykresie
         plt.scatter(pca_data[:, 0], pca_data[:, 1],
                     c=target, edgecolor='none', alpha=0.5,
                     cmap=plt.cm.get_cmap('Accent', 10))
         plt.xlabel('component 1component 1')
         plt.ylabel('component 2')
         plt.title('Dimmensionality reduction using SVD')
         plt.colorbar();
```

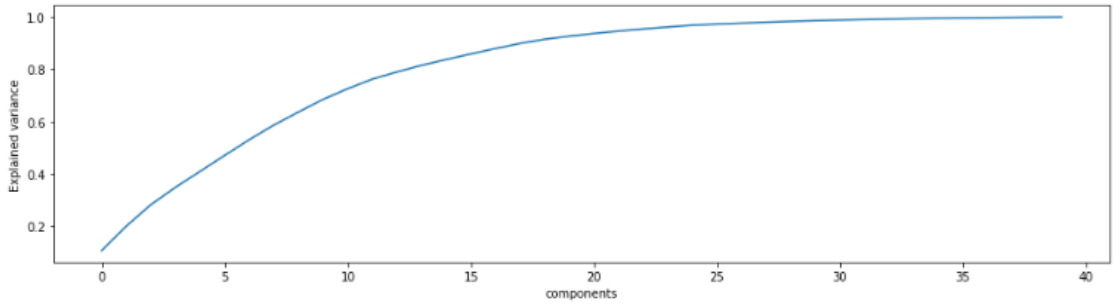


Rysunek 21. Wykres dekompozycji danych metodą PCA.

PCA opisuje liniową redukcje wymiarów za pomocą dekompozycji wartości pojedynczej danych w celu rzutowania ich na przestrzeń o niższych wymiarach. Dane wejściowe są wyśrodkowane, ale nie są skalowane dla każdej funkcji przed zastosowaniem SVD.

Wynik wariancji z wykorzystaniem PCA:

```
In [45]: #Explained Variance -> SVD for Online news popularity
pca = PCA().fit(pca_data)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('components')
plt.ylabel('Explained variance');
```



Rysunek 22. Przebieg zmian wartości współczynnika 'explained variance' wraz ze wzrostem liczby wymiarów.

Z wykresu można odczytać jak zmienia się wartość współczynnika ‘explained variance’ wraz ze wzrostem liczby wymiarów na naszych danych. Można zauważyć, że wartość tego współczynnika rośnie wprost proporcjonalnie do ilości wymiarów.

W kolejnym kroku podejmujemy dzielenie danych na uczące i testowe:

```
In [46]: #DZIELENIE DANYCH NA MODELE UCZĄCE I TESTOWE

from sklearn.model_selection import train_test_split
data_train, data_test, target_train, target_test = train_test_split(pca_data, target, test_size = 0.10)
print("Training dataset:")
print("data_train:", data_train.shape)
print("target_train:", target_train.shape)
print("\nTesting dataset:")
print("data_test:", data_test.shape)
print("target_test:", target_test.shape)
```

```
Training dataset:
data_train: (31591, 40)
target_train: (31591,)

Testing dataset:
data_test: (3511, 40)
target_test: (3511,)
```

Rysunek 23. Implementacja - podział danych na modele uczące i testowe.

Różnica jaką możemy zauważyć odnosząc się do poprzedniego przykładu regresji liniowej zawartym w podrozdziale 2.3.1, to nowy argument ‘pca_data’.

```
In [47]: #UCZENIE MODELU
         from sklearn.linear_model import LinearRegression

         linear_regression = LinearRegression()
         linear_regression.fit(data_train, target_train)

Out[47]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [48]: #PRZEWIDYWANIE
         prediction_linear = linear_regression.predict(data_test)

In [49]: df_someXdata = pd.DataFrame(linear_regression.predict(data_train),list(target_train) )
         df_someXdata.reset_index(level=0, inplace=True)
         df_someXdata_LR = df_someXdata.rename(index=str, columns={"index": "Actual shares", 0: "Predicted shares"})
         df_someXdata_LR.head(10)
```

Rysunek 24. Implementacja - uczenie modelu oraz przewidywanie targetu.

Wynik predykcji zawarty w tabeli:

Out[49]:

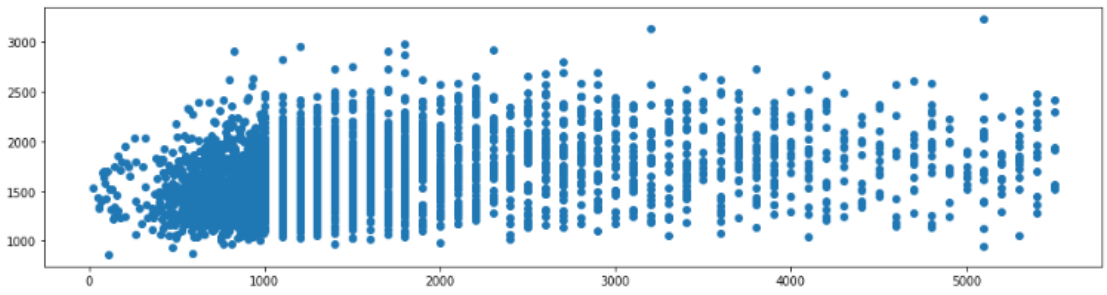
	Actual shares	Predicted shares
0	466	1498.580885
1	1100	1294.778547
2	659	1716.875293
3	927	1807.571933
4	2500	2621.215864
5	735	1784.359111
6	921	1959.271025
7	949	1800.630502
8	4600	1972.060209
9	3700	1481.478535

Rysunek 25. Tabela z wynikami przewidywania wyuczonego modelu.

Wykres predykcji PCA:

```
In [50]: #WYKRES PREDYKCJI
         plt.scatter(target_test, prediction_linear)
         print("Wykres predykcji")
```

```
Wykres predykcji
```



Rysunek 26. Wykres predykcji - regresja liniowa z redukcją wymiarowości dla testowanych danych.

```
In [51]: #Ocena jakości metodą średniego błędu kwadratowego
         from sklearn.metrics import mean_squared_error
         mse =  mean_squared_error(target_test, prediction_linear)
         rmse = np.sqrt(mse)
         print("Średni błąd kwadratu wyuczonego modelu: %.2f" %  mse)
         print("Pierwiastek średniego błędu kwadratowego: %.2f" %  rmse)

         Średni błąd kwadratu wyuczonego modelu: 1078854.93
         Pierwiastek średniego błędu kwadratowego: 1038.68
```

```
In [52]: from sklearn.metrics import r2_score
         print('Variance score: %.2f' % r2_score(target_test, linear_regression.predict(data_test)))

         #Wniosek -> Wynik gorszy niż w przypadku czystej regresji

         Variance score: 0.09
```

Rysunek 27. Średni błąd kwadratowy, wynik dokładności przewidywania.

Wyliczając średni błąd kwadratu, widzimy że zwiększył się on w porównaniu do czystej regresji liniowej o około 23 000, tym samym wskazując mniejszą dokładność przewidywania modelu równą 9%.

Truncated SVD

W naszym projekcie postanowiliśmy również poddać nasz model redukcji wymiarów za pomocą skróconego SVD (truncated SVD). Ten transformator dokonuje liniowej redukcji wymiarowości za pomocą skróconego rozkładu wartości pojedynczej (SVD). W przeciwieństwie do PCA, estymator nie centruje danych przed obliczeniem rozkładu wartości osobliwych. Oznacza to, że może efektywnie pracuje z rzadkimi macierzami.

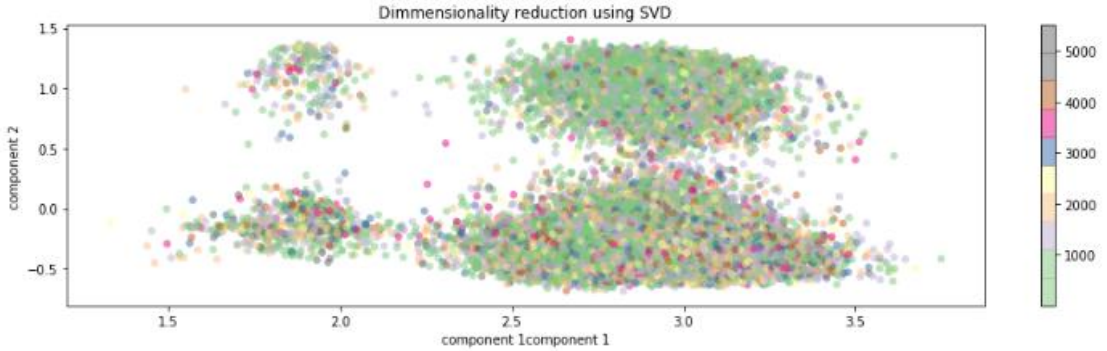
Zaczynając od początku, importujemy metodę TruncatedSVD z biblioteki sklearn. Następnie używamy jej na naszym modelu.

```
In [54]: # Redukcja wymiarowości metodą SVD Truncated
         from sklearn.decomposition import TruncatedSVD
         svd = TruncatedSVD(n_components=40)
         svd_data = svd.fit(data).transform(data)
         svd_data

Out[54]: array([[ 2.62380645e+00, -3.59791695e-01,  1.80595976e-02, ...,
                  -3.45782596e-02,  9.76171699e-03,  3.36547341e-02],
                [ 2.64174482e+00, -4.09553008e-01,  3.46061020e-01, ...,
                  -5.09107745e-02,  3.46355215e-02,  3.92974925e-02],
                [ 2.52533696e+00,  1.56369873e-02, -3.75860086e-01, ...,
                  -3.52499749e-02,  4.26108337e-02,  1.47365666e-02],
                ...,
                [ 2.64297466e+00, -2.43103717e-01, -3.17757782e-01, ...,
                   1.92722098e-03, -2.95924072e-03,  1.72853050e-02],
                [ 2.79065900e+00,  1.26401592e+00,  6.93369283e-02, ...,
                  -2.88794206e-03, -7.21435960e-03,  1.49900220e-02],
                [ 2.93083338e+00, -4.60547368e-01, -7.12926682e-01, ...,
                   7.90801737e-02, -2.07257920e-02, -4.40636724e-02]])
```

Rysunek 28. Redukcja wymiarowości metodą SVD Truncated.

Wizualizacja wykresu, po użyciu metody ‘TruncatedSVD’:

```
In [55]: #Wizualizacja na wykresie
plt.scatter(svd_data[:, 0], svd_data[:, 1],
            c=target, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('Accent', 10))
plt.xlabel('component 1component 1')
plt.ylabel('component 2')
plt.title('Dimmensionality reduction using SVD')
plt.colorbar();
```



Rysunek 29. Wykres przedstawiający redukcje wymiarowości metodą Truncated SVD badanego zestawu danych.

Trenowanie zestawu danych:

```
In [56]: from sklearn.model_selection import train_test_split
data_train, data_test, target_train, target_test = train_test_split(svd_data, target, test_size = 0.10)
print("Training dataset:")
print("data_train:", data_train.shape)
print("target_train:", target_train.shape)
print("\nTesting dataset:")
print("data_test:", data_test.shape)
print("target_test:", target_test.shape)

Training dataset:
data_train: (31591, 40)
target_train: (31591,)

Testing dataset:
data_test: (3511, 40)
target_test: (3511,)
```

Rysunek 30. Implementacja - uczenie modelu z wykorzystaniem redukcji wymiarowości metodą TruncatedSVD.

Uczenie oraz przywidywanie dokładności modelu:

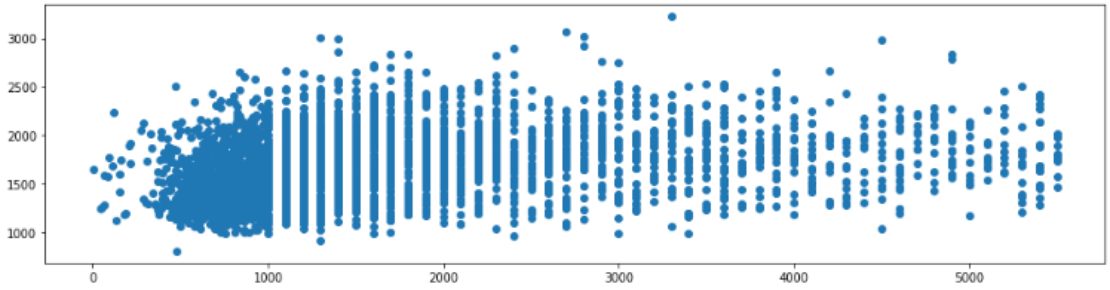
```
In [57]: #UCZENIE MODELU
from sklearn.linear_model import LinearRegression

linear_regression = LinearRegression()
linear_regression.fit(data_train, target_train)

Out[57]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [58]: #PRZEWIDYWANIE
prediction_linear = linear_regression.predict(data_test)

In [59]: df_someXdata = pd.DataFrame(linear_regression.predict(data_train),list(target_train) )
df_someXdata.reset_index(level=0, inplace=True)
df_someXdata_LR = df_someXdata.rename(index=str, columns={"index": "Actual shares", 0: "Predicted shares"})
df_someXdata_LR.head(10)
```

Rysunek 31. Implementacja - uczenie modelu oraz przewidywanie regresji liniowej.

Wykres predykcji:

```
In [60]: #WYKRES PREDYKCJI
plt.scatter(target_test, prediction_linear)
print("Wykres predykcji")
```

```
Wykres predykcji
```



Rysunek 32. Wykres predykcji modelu opartego na regresji liniowej z wykorzystaniem redukcji wymiarowości metody Truncated SVD.

Średni błąd kwadratu:

```
In [61]: #Ocena jakości metodą średniego błędu kwadratowego
from sklearn.metrics import mean_squared_error
mse =  mean_squared_error(target_test, prediction_linear)
rmse = np.sqrt(mse)
print("Średni błąd kwadratu wyuczonego modelu: %.2f" %  mse)
print("Pierwiastek średniego błędu kwadratowego: %.2f" %  rmse)
```

```
Średni błąd kwadratu wyuczonego modelu: 1154251.68
Pierwiastek średniego błędu kwadratowego: 1074.36
```

```
In [62]: from sklearn.metrics import r2_score
print('Variance score: %.2f' % r2_score(target_test, linear_regression.predict(data_test)))

#Wniosek -> Wynik gorszy niż w przypadku czystej regresji
```

```
Variance score: 0.08
```

Rysunek 33. Średni błąd kwadratu modelu opartego na regresji liniowej z wykorzystaniem redukcji wymiarowości Truncated SVD.

Po poddaniu naszego modelu ocenie jakości średniego błędu kwadratowego widzimy, iż błąd ten zwiększył się około 70 000. Analizując poprzednie przykłady możemy wywnioskować że im większy średni błąd kwadratowy tym mniejsza będzie dokładność przewidywania. Do tej pory jest to najsłabszy pomiar (Variance score: 8%) biorąc pod uwagę czystą regresję liniową, regresję liniową z zastosowaniem redukcji wymiarowości PCA oraz truncated SVD.

2.3.3. Trenowanie modelu – Sieć neuronowa

Do naszych badań wykorzystaliśmy również sieć neuronową. Pracując na zestawie danych ‘Online News Popularity’, przeprowadziliśmy szkolenie modelu. Poniższy fragment kodu prezentuje wykorzystanie wielowarstwowego regresoru Perceptron (MLPRegressor). Model ten optymalizuje straty kwadratowe za pomocą LBFGS lub stochastycznego spadku.

```
In [10]: from sklearn.neural_network import MLPRegressor
         from sklearn.metrics import mean_squared_error

         news_network = MLPRegressor(solver='adam', alpha=1e-5,
                             hidden_layer_sizes=(50,50 ),verbose=True, random_state=1, max_iter=5000, tol=1e-7)
         news_network.fit(data_train, target_train)

         mse =  mean_squared_error(target_test, news_network.predict(data_test))
         rmse = np.sqrt(mse)
         print("Średni błąd kwadratu wyuczonego modelu: %.2f" %  mse)
         print("Pierwiastek średniego błędu kwadratowego: %.2f" %  rmse)

         Iteration 1, loss = 1858179.75508696
         Iteration 2, loss = 826665.20848357
         Iteration 3, loss = 596078.43239721
         Iteration 4, loss = 585839.34981843
         Iteration 5, loss = 577875.99005682
         Iteration 6, loss = 572023.60435248
         Iteration 7, loss = 567593.46699285
         Iteration 8, loss = 564494.05464245
         Iteration 9, loss = 562147.37672121
         Iteration 10, loss = 560490.01002522
         Iteration 11, loss = 559253.49822100
         Iteration 12, loss = 558137.31475145
         Iteration 13, loss = 557223.36019001
         Iteration 14, loss = 556291.34908090
         Iteration 15, loss = 555555.07000675
         Iteration 16, loss = 554890.10070934
         Iteration 17, loss = 554503.79302282
         Iteration 18, loss = 553846.89549293
         Iteration 19, loss = 553546.48908717
         Iteration 20, loss = 553080.10023478
```

Rysunek 34. Implementacja – wielowarstwowy regresom perceptron wykorzystany w Siecie neuronowej do trenowania modelu.

Atrybuty jakie przyjmuje MLPRegressor to:

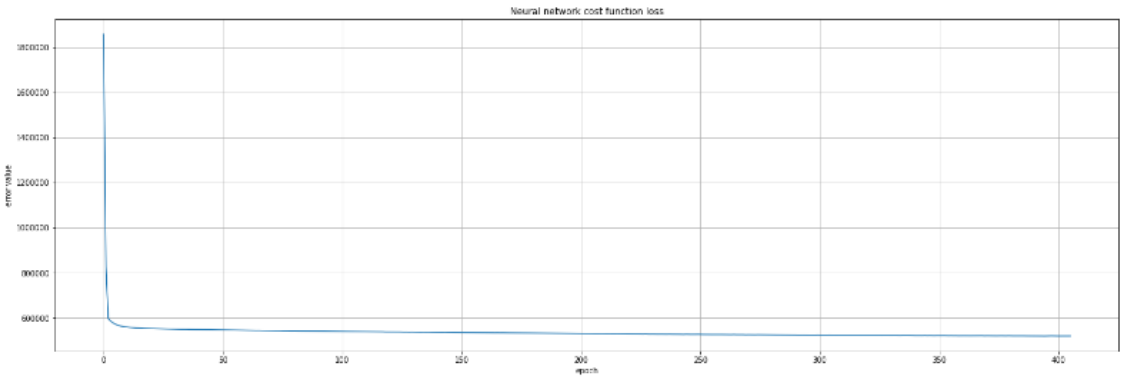
- Hidden_layer_size – określa rozmiar warstw; I-ty element reprezentuje liczbę neuronów w i-tej ukrytej warstwie
- Solver – służy do optymalizacji masy, w naszym przypadku zastosowaliśmy wersję ‘adam’, która odnosi się do stochastycznego optymalizatora opartego na gradiencie
- Alpha – jest to parametr regularyzacji
- Max_iter - Maksymalna liczba iteracji. Solver wykonuje iterację do konwergencji (określonej przez „tol”) lub tej liczby iteracji. W przypadku solverów stochastycznych („sgd”, „adam”) należy pamiętać, że określa to liczbę epok (ile razy zostanie użyty każdy punkt danych), a nie liczbę kroków gradientu.
- Random_state – określa generowanie liczb losowych dla inicjalizacji wag i obciążenia
- Verbose - określa, czy drukować komunikaty o postępach na standardowe wyjście.

Efekt końcowy trenowanego modelu zwraca wynik lepszy, niż wcześniejsze wyniki modelowane na innych wariantach.

```
In [11]: from sklearn.metrics import r2_score
         print('Variance score: %.2f' % r2_score(target_test, news_network.predict(data_test)))

         Variance score: 0.14
```

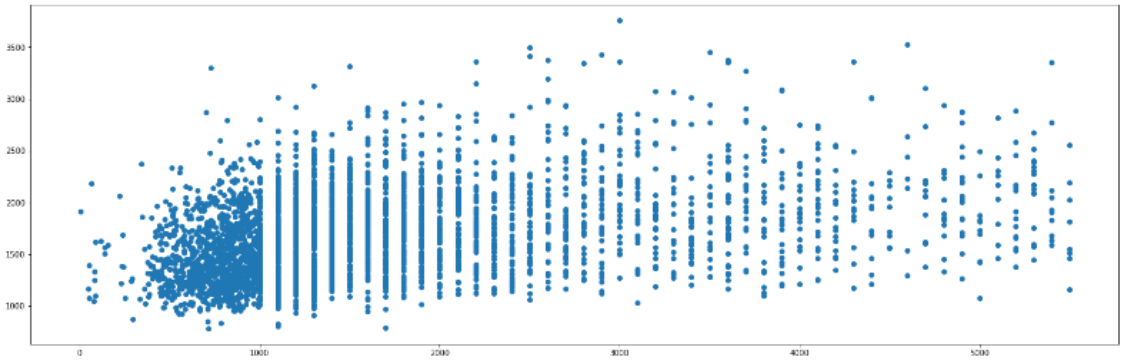
Rysunek 35. Wynik wariancji sieci neuronowej.

```
In [14]: import matplotlib.pyplot as plt
         plt.plot(news_network.loss_curve_)
         plt.title('Neural network cost function loss')
         fig_size = plt.rcParams["figure.figsize"]
         fig_size[0] =25.0
         fig_size[1] = 8.0
         plt.xlabel('epoch'); plt.ylabel('error value'); plt.grid();
```



Rysunek 36. Wykres utraty funkcji kosztu sieci neuronowej.

```
In [15]: #WYKRES PREDYKCJI
         plt.scatter(target_test, news_network.predict(data_test))
         print("Wykres predykcji")

         Wykres predykcji
```



Rysunek 37. Wykres predykcji dla sieci neuronowej.

```
In [16]: df_someXdata = pd.DataFrame(news_network.predict(data_train),list(target_train) )
         df_someXdata.reset_index(level=0, inplace=True)
         df_someXdata_LR = df_someXdata.rename(index=str, columns={"index": "Actual shares", 0: "Predicted shares"})
         df_someXdata_LR.head(11)
```

Out[16]:

	Actual shares	Predicted shares
0	4500	2094.889819
1	1100	1268.351038
2	1500	2201.186220
3	1800	1200.547584
4	436	1469.750528
5	889	1862.721503
6	424	1243.337618
7	1800	1296.046838
8	1300	1959.029713
9	1200	2099.698434
10	1700	1433.122223

Rysunek 38. Wytrenowany model przedstawiający wynik przewidywania trenowanego przykładu.

2.3.4. Trenowanie modelu – Sieć neuronowa z redukcją wymiarowości

Ostatecznym wariantem jaki postanowiliśmy przetestować jest wytrenowanie modelu na sieci neuronowej z redukcją wymiarowości. Po wykonaniu wstępnych czynności, czyli usunięciu wartości odstających, ustaleniu wartości wejściowych, celu oraz wykonaniu standaryzacji danych przystąpiliśmy do redukcji wymiarowości metodą PCA.

```
In [7]: #Siec neuronowa z metodą PCA redukcji wymiarowości
        from sklearn.decomposition import PCA
        from sklearn.preprocessing import MinMaxScaler

        pca = PCA(n_components=40) #Metoda PCA
        pca_data = pca.fit(data).transform(data)
        pca_data
```

```
Out[7]: array([[-0.3555699 ,  0.00866528,  1.23732807, ...,  0.00342306,
                 0.01260056,  0.01503903],
                [-0.41505111,  0.34104306,  0.74493274, ..., -0.00501034,
                 -0.00948469,  0.01465762],
                [ 0.03223653, -0.39460297, -0.28554368, ...,  0.0114035 ,
                 -0.016694  , -0.00609487],
                ...,
                [-0.227935  , -0.34363471, -0.17989547, ...,  0.00802174,
                  0.00784163,  0.01420245],
                [ 1.26219197,  0.09502245,  0.18772458, ..., -0.00180843,
                  0.00843976,  0.01342379],
                [-0.45036292, -0.71759545, -0.38607578, ...,  0.07241938,
                  0.02678224, -0.04323739]])
```

Rysunek 39. Redukcja wymiarowości PCA, siec neuronowa.

W kolejnym kroku podzieliliśmy model na dane uczące i testowe.

```
In [8]: #DZIELENIE DANYCH NA MODELE UCZĄCE I TESTOWE

from sklearn.model_selection import train_test_split
data_train, data_test, target_train, target_test = train_test_split(pca_data, target, test_size = 0.10)
print("Training dataset:")
print("data_train:", data_train.shape)
print("target_train:", target_train.shape)
print("\nTesting dataset:")
print("data_test:", data_test.shape)
print("target_test:", target_test.shape)
```

```
Training dataset:
data_train: (31591, 40)
target_train: (31591,)

Testing dataset:
data_test: (3511, 40)
target_test: (3511,)
```

Rysunek 40. Implementacja przedstawiająca podział danych na modele uczące i testowe.

Dla wszystkich trenowanych modeli przyjmujemy identyczny stosunek danych uczących i testowych, w wymiarze 90% i 10%.

```
In [9]: from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error

news_network = MLPRegressor(solver='adam', alpha=1e-5,
                    hidden_layer_sizes=(50,50 ),verbose=True, random_state=1, max_iter=5000, tol=1e-7)
news_network.fit(data_train, target_train)

mse =  mean_squared_error(target_test, news_network.predict(data_test))
rmse = np.sqrt(mse)
print("Średni błąd kwadratu wyuczonego modelu: %.2f" %  mse)
print("Pierwiastek średniego błędu kwadratowego: %.2f" %  rmse)
```

```
Iteration 1464, loss = 450136.61972461
Iteration 1465, loss = 449789.11785986
Iteration 1466, loss = 449783.98678925
Iteration 1467, loss = 449837.76008478
Iteration 1468, loss = 449850.28468998
Iteration 1469, loss = 449524.26017587
Iteration 1470, loss = 449904.41307314
Iteration 1471, loss = 449643.09476275
Iteration 1472, loss = 449764.64335942
Iteration 1473, loss = 449540.78815379
Iteration 1474, loss = 449742.16576162
Iteration 1475, loss = 449698.66162437
Iteration 1476, loss = 449905.80296011
Iteration 1477, loss = 449637.81905185
Iteration 1478, loss = 449692.37567920
Iteration 1479, loss = 449546.12023221
Iteration 1480, loss = 449672.17796285
Training loss did not improve more than tol=0.000000 for 10 consecutive epochs. Stopping.
Średni błąd kwadratu wyuczonego modelu: 1153415.62
Pierwiastek średniego błędu kwadratowego: 1073.97
```

Rysunek 41. MLPRegressor.

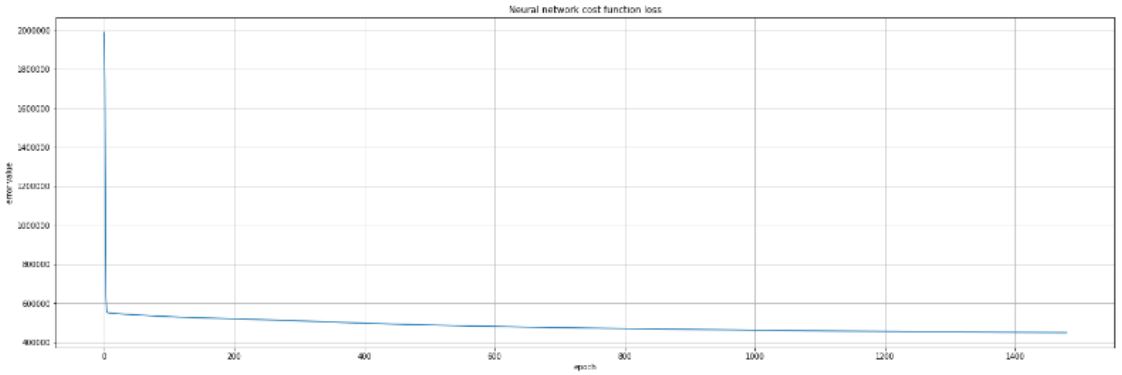
Metoda MLPRegressor przyjmuje identyczne atrybuty jak w wariancie szkolonym w podrozdziale 2.3.3. Średni błąd kwadratu wyuczonego modelu jest niemalże identyczny jak we wspomnianym wariancie, lecz wynik R2 jest zdecydowanie niższy niż w poprzednich modelach i wynosi 4%. Świadczy to o tym, że metoda PCA nie najlepiej współpracuje na naszych danych w modelu sieci neuronowej.

```
In [10]: from sklearn.metrics import r2_score
print('Variance score: %.2f' % r2_score(target_test, news_network.predict(data_test)))
```

```
Variance score: 0.04
```

Rysunek 42. Wynik wariancji modelu sieci neuronowej z redukcją wymiarowości metodą PCA.

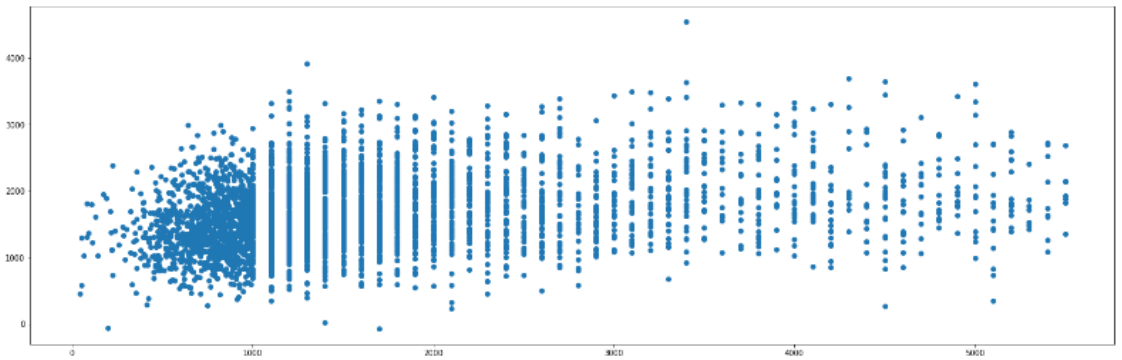
In [19]:
```
import matplotlib.pyplot as plt
plt.plot(news_network.loss_curve_)
plt.title('Neural network cost function loss')
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] =25.0
fig_size[1] = 8.0
plt.xlabel('epoch'); plt.ylabel('error value'); plt.grid();
```



Rysunek 43. Wykres utraty funkcji kosztu sieci neuronowej z redukcją wymiarowości.

In [21]:
```
#WYKRES PREDYKCJI
plt.scatter(target_test, news_network.predict(data_test))
print("Wykres predykcji")
```

```
Wykres predykcji
```



Rysunek 44. Wykres predykcji sieci neuronowej z redukcją wymiarowości.

In [22]:
```
df_someXdata = pd.DataFrame(news_network.predict(data_train),list(target_train) )
df_someXdata.reset_index(level=0, inplace=True)
df_someXdata_LR = df_someXdata.rename(index=str, columns={"index": "Actual shares", 0: "Predicted shares"})
df_someXdata_LR.head(11)
```

Out[22]:

	Actual shares	Predicted shares
0	1200	1383.360436
1	1300	2140.884167
2	2300	1541.378543
3	666	1717.060181
4	2700	2275.377166
5	672	1721.040722
6	1300	1518.914322
7	926	422.230086
8	863	1676.597112
9	3200	2169.052921
10	3700	2585.573489

Rysunek 425. Wyniki przewidywania dla sieci neuronowej z redukcją wymiarowości.

W projekcie rozważyliśmy również sieć neuronowa z metodą Truncated SVD.

```
In [23]: #-----------------------------------------------------------------
         #Sieć neuronowa z metodą Truncated SVD
         |
         from sklearn.decomposition import TruncatedSVD
         svd = TruncatedSVD(n_components=40)
         svd_data = svd.fit(data).transform(data)
         svd_data

Out[23]: array([[ 2.62380645e+00, -3.59791695e-01,  1.80595976e-02, ...,
                  -3.45782800e-02,  9.76181101e-03,  3.36542123e-02],
                [ 2.64174482e+00, -4.09553008e-01,  3.46061020e-01, ...,
                  -5.09107673e-02,  3.46354213e-02,  3.92973989e-02],
                [ 2.52533696e+00,  1.56369873e-02, -3.75860086e-01, ...,
                  -3.52500294e-02,  4.26108219e-02,  1.47346054e-02],
                ...,
                [ 2.64297466e+00, -2.43103717e-01, -3.17757782e-01, ...,
                   1.92720297e-03, -2.95914050e-03,  1.72849506e-02],
                [ 2.79065900e+00,  1.26401592e+00,  6.93369283e-02, ...,
                  -2.88792439e-03, -7.21438615e-03,  1.49905705e-02],
                [ 2.93083338e+00, -4.60547368e-01, -7.12926682e-01, ...,
                   7.90801044e-02, -2.07259006e-02, -4.40663438e-02]])
```

Rysunek 46. Dekompozycja danych metodą Truncated SVD dla sieci neuronowej.

W przypadku tym pojawiła się różnica w wyniku dokładności modelu, który wyniósł:

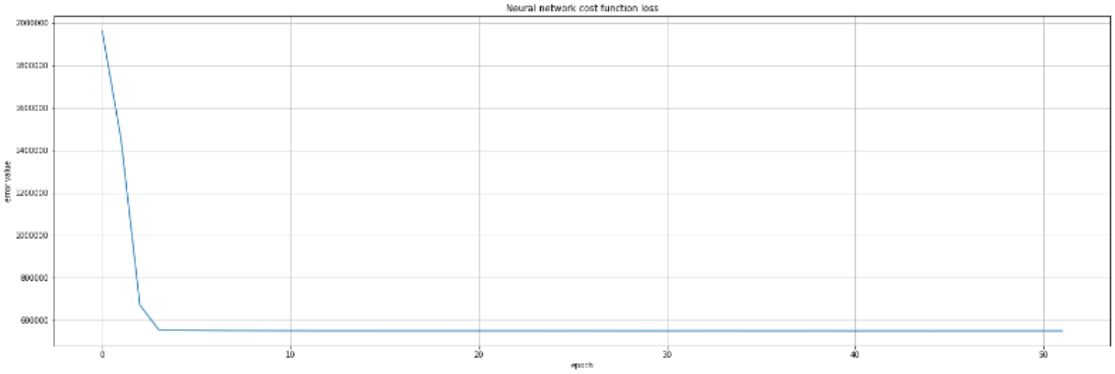
- variance score: 0.10

Natomiast w przykładzie uczenia modelu poprzez sieć neuronową otrzymaliśmy dokładność na poziomie 0.14.

```
In [26]: from sklearn.metrics import r2_score
         print('Variance score: %.2f' % r2_score(target_test, news_network.predict(data_test)))

         Variance score: 0.10
```

Rysunek 47. Wynik wariancji modelu sieci neuronowej z redukcją wymiarowości metodą Truncated SVD.

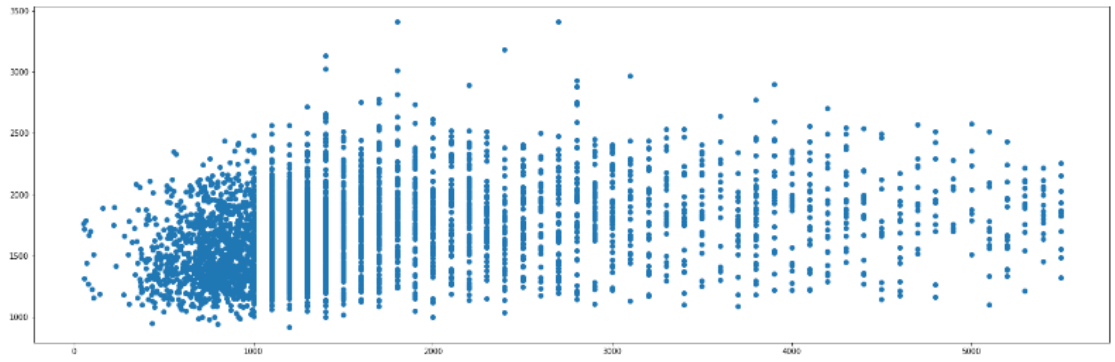
```
In [28]: import matplotlib.pyplot as plt
         plt.plot(news_network.loss_curve_)
         plt.title('Neural network cost function loss')

         plt.xlabel('epoch'); plt.ylabel('error value'); plt.grid();
```



Rysunek 48. Wykres utraty funkcji kosztu sieci neuronowej z metodą Truncated SVD.

```
In [29]: #WYKRES PREDYKCJI
plt.scatter(target_test, news_network.predict(data_test))
print("Wykres predykcji")
```

```
Wykres predykcji
```



Rysunek 49. Wykres predykcji dla sieci neuronowej z metodą Truncated SVD.

Out[30]:

	Actual shares	Predicted shares
0	1100	1362.050707
1	4000	1858.232394
2	1800	1499.385193
3	1500	2288.851678
4	919	1060.608535
5	1700	1868.228465
6	786	2101.142181
7	3000	2097.133840
8	2300	1717.562077
9	879	1530.113559
10	1500	1594.481680

Rysunek 50. Wyniki przewidywania modelu sieci neuronowej z wykorzystaniem metody Truncated SVD.

3. Analiza otrzymanych wyników

Otrzymaliśmy następujące wyniki z wszystkich utworzony modeli w projekcie:

- Regresja liniowa -> 12% dokładności.

  Celem naszego projektu było utworzenie modelu regresji liniowej dla danych przewidywania ilości udostępnień artykułów w Internecie. Wynik R2, który jest najważniejszym wynikiem dla nas, przedstawia w formie procentowej, jak bardzo dokładny jest utworzony model. Niestety po przeprowadzonych optymalizacjach opisanych w rozdziale drugim, nie udało się osiągnąć zadowalającego wyniku R2 w modelu regresji liniowej. W związku z tym zdecydowaliśmy się podjąć następny krok w celu uzyskania wyższego wyniku. Tym krokiem jest zastosowanie redukcji wymiarowości.

- Regresja liniowa z zastosowaniem redukcji wymiarowości (40 wymiarów z 58) -> PCA 9%, TruncatedSVD 8% dokładności.

  Redukcję wymiarowości zastosowaliśmy w dwóch metodach: PCA i TruncatedSVD, aby mieć pewność, że niczego nie przeoczyliśmy. W każdej z tych metod zastosowaliśmy redukcję do 40 wymiarów, gdyż na niej uzyskiwaliśmy najlepsze wyniki. Niestety wynik R2 jest gorszy niż w przypadku czystej regresji liniowej. Wniosek z tej operacji nasuwa się taki, że redukcja wymiarowości dla naszej bazy danych nie daje pożądanych rezultatów. W związku z tym użyliśmy model sieci neuronowej. Jest to dużo bardziej zaawansowana metoda uczenia niż regresja liniowa. Wykorzystuje bardziej złożone mechanizmy, przez co między innymi proces uczenia trwa dłużej, ale można liczyć na lepsze rezultaty.

- Sieć neuronowa -> 14% dokładności.

  Po wytrenowaniu modelu sieci neuronowej na warstwach (50,50) otrzymaliśmy najlepszy wynik R2, który wynosi 14%. Jest to dalej wynik poniżej oczekiwań. Dla upewnienia się, przeprowadziliśmy jeszcze redukcje wymiarowości na ten model.

- Sieć neuronowa z zastosowaniem redukcji wymiarowości (40 wymiarów z 58) -> PCA 4%, TruncatedSVD 10% dokładności.

  Podobnie jak to było w przypadku regresji liniowej z zastosowaniem redukcji wymiarowości, tak samo w przypadku zastosowania tej operacji na sieci neuronowej wynik jest gorszy. Oznacza to, że redukcja wymiarowości używana w naszych modelach uczących nie poprawia stopnia dokładności modeli.

Z analizy wszystkich modeli wynika, że najlepszym modelem dla naszego datasetu jest model sieci neuronowej, który otrzymał wynik R2 w wysokości 14%.

4. Wnioski

1. Najlepszy wynik uzyskano z modelu sieci neuronowej (14%), choć dalej jest to wynik poniżej oczekiwań.
2. Pomimo zastosowania wielu optymalizacji danych, maksymalny wynik wytrenowania uzyskany z wszystkich utworzonych modeli to 14%, co jest wynikiem zdecydowanie zbyt niskim.
3. Bez optymalizacji badanego zestawu danych, wynik wariancji trenowanego modelu wynosił około 3%. W porównaniu do najlepszego modelu uczenia maszynowego, różnica między nimi wyniosła 10 punktów procentowych.
4. Ogromna liczba różnych danych na których przeprowadzaliśmy nasze badania sprawia, że wytrenowanie modelu na poziomie 60-70% dokładności, wymagało by jeszcze większej optymalizacji zestawu danych.
5. Model sieci neuronowej z zastosowaniem redukcji wymiarowości szkolił się najdłuższą ilość czasu.
6. Redukcja wymiarowości nie poprawia wyniku wyuczenia modelu.
7. Baza danych, na podstawie której został utworzony projekt sprawia, że ciężko jest utworzyć dokładny model uczenia maszynowego.
8. W sieciach neuronowych istnieje możliwość uzyskania trochę lepszego wyniku odpowiednio określając ilość warstw oraz parametrów uczących w metodzie MLPRegressor.